基于软件基因的 Android 恶意软件检测与分类

韩 金，单 征，赵炳麟，孙文杰

(数学工程与先进计算国家重点实验室，郑州 450001)

摘 要：随着移动互联网的发展，针对 Android 平台的恶意代码呈现急剧增长。而现有的 Android 恶意代码分析方法多聚焦于基于特征对恶意代码的检测，缺少统一的系统化的分析方法，且少有对恶意代码分类的研究。基于这种现状，提出了恶意软件基因的概念，以包含功能信息的片段对恶意代码进行分析；基于 Android 平台软件的特点，通过代码段和资源段分别提取了软件基因，其中代码段基因基于 use-def 链（使用-定义链）进行形式化。此外，分别提出了基于恶意软件基因的检测框架和分类框架，通过机器学习中的支持向量机对恶意软件基因进行学习，有较高的检测率和分类正确率，其中检测召回率达到了 98.37%，验证了恶意软件基因在分析同源性中的作用。

关键词：Android 安全；恶意软件基因；use-def 链；检测；分类

中图分类号：TP309.5 doi: 10.3969/j.issn.1001-3695.2018.01.0007

Detection and classification of Android malware based on malware gene

Han Jin, Shan Zhen, Zhao Binglin, Sun Wenjie

(*State Key Laboratory of Mathematical Engineering & Advanced Computing*, *Zhengzhou* 450001, *China*)

Abstract: With the development of mobile internet, malicious code for Android platform has increased dramatically. And face up to the mount of Android malware, the current analyzing methods are focusing on the characteristic-based detecting, which is lack of a uniform systematic analyzing and classifying method. To resolve this status, this paper proposed the definition of Android malware gene to analyzing malware via binary sequence including function and information. And based on the characteristics of Android applications, this paper extract software gene from code fragment and resource fragment. Therein, the code fragment gene is a kind of formalization of use-def chains. Moreover, this paper proposed a detecting framework and a classifying framework based on malware gene. And this paper utilized a machine learning method, support vector machine(SVM) , in the frameworks. In evaluation, the detecting rate and classification correct rate are both high in those frameworks, with a recall rate of 98.37%. It proves the effect of malware gene in analyzing the homology of Android malicious code.

Key words: Android security; malware gene; use-def chains; detection; classification

0 引言

自从 2007 年面世起，Android 系统逐渐成为了在移动终端应用最广泛的操作系统。据国际著名研究分析机构 Gartner 公司[1]的报告，在 2016 年第四季度，搭载 Android 操作系统的手机出货量为 2.94 亿，约占 84.1%的智能手机市场份额。并且，相比于个人电脑，移动设备与用户的隐私信息和财产安全联系更加密切。此外，由于系统的开源性与市场的开放性，Android 操作系统更加容易被恶意软件利用。因此，Android 平台已经愈发成为黑色产业的目标。根据阿里巴巴在 2015 年的报告[2]显示，18%的 Android 设备曾干扰过恶意软件，95%的 Android 应用存在仿冒应用，当年度阿里聚安全共检测到了 3 亿个 Android 平台的恶意软件，而这些恶意软件可以被分为多种。其中，以恶意软件的行为来看，流氓行为类以 52.4%的比率占据首位，这类恶意软件的行为包括匿名弹窗、恶意推送广告、私自下载软件等。而恶意扣费、隐私窃取、短信劫持、盗版软件、系统破坏和诱骗欺诈等类型的恶意软件则以 22.3%、13.0%、6.4%、3.2%、 1.4%、 1.2%的比例位居其后。

目前，恶意软件的检测方法主要分为静态分析与动态分析，而对于大规模的恶意代码分析，静态分析更加快捷方便。

基于权限的检测工具 Kirin[3]是早期有代表性的静态分析方法之一，它通过定义的一组规则识别危险的权限组合，其误

收稿日期：2018-01-09；修回日期：2018-02-26

作者简介：韩金（1993-），男，安徽五河人，硕士研究生，主要研究方向为 Android 恶意代码分析（hanjin1993@outlook.com）；单征（1977-），男，教授，博士，主要研究方向为信息安全、恶意代码分析；赵炳麟（1990-），男，博士研究生，主要研究方向为信息安全、恶意代码分析；孙文杰（1994 年-），男，硕士研究生，主要研究方向为信息安全.

报率较高。与之类似，Zarni[4]等人也提出了一种基于权限的恶意代码检测框架，而基于其他特征的静态分析解决方案还有如下几种：

Fuchs 等人设计的 SCanDroid[5]，使用数据流分析方法进行静态分析，从应用的 AndroidManifest.xml 文件中提取权限，然后自动检测应用的数据流是否与这些权限一致。类似地，Wu 等人设计的 DroidMat[6]系统也是在配置文件 AndroidManifest.xml 的基础上进行实验，同时提取了 API 调用序列。Chan 等人设计的 DroidChecker[7] 使用控制流图搜索和污点分析方法自动分析可能的敏感数据泄露路径。Lu 等人提出的 Chex[8]，是基于组件的静态分析方法。Kim 等人提出的 ScanDal[9]以 Dalvik 虚拟机字节码为输入，检测 Android 应用程序的私密信息泄露。而国内的李挺等人[10]也提出了一种基于 Dalvik 指令的恶意代码检测方法，该方法无须对软件进行反编译。但这些工具仍然有其局限性：有的未进行组件间通信分析，有的未考虑 Android 的事件驱动编程范式。针对这些问题，Cui 等人提出的 CoChecker[11]工具基于 Android API 调用将泄露路径分为两类的方法:权能泄露路径和敏感数据泄露路径。此外 Yu 等人设计的 Apposcopy[12]系统和 Rastogi 等人开发的 DroidChameleon[13] 都是通过基于语义的静态分析方法来分析 Android 恶意软件。

而以上这些研究，多是针对恶意代码之间的相似性而非同源性，且研究的成果也多是以恶意代码检测系统为主，对于恶意代码分类鲜有研究。而目前的 Android 平台恶意代码分类标准混乱，基本都是以恶意软件的行为进行分类，如 Zhou 等人[14]将恶意软件分为：恶意安装、恶意运行、恶意代码载荷和权限使用等类别。而对于反病毒引擎来说，一般会根据恶意代码家族对其进行标记，然而，不同的反病毒引擎对恶意代码的标志规则不一。AVClass[15]对反病毒引擎的标志进行了处理与综合，是目前为恶意代码做家族标志较好的系统，但易受个别被多种杀毒软件使用的反病毒引擎影响，也没有跳出反病毒引擎命名的掣肘。

面对如此数量庞大、更新迅速的 Android 平台恶意软件，对其进行有效地检测与分类是很有必要的，希望对 Android 平台的恶意软件提取出软件基因，基于恶意软件基因进行数据挖掘和机器学习，对大数据的样本进行训练，形成模型，以解决对日益发展的 Android 恶意样本进行检测和分类的任务。本文结合语义特征与语法特征系统化地抽象了 Android 恶意软件的功能信息，并将其定义为 Android 恶意软件基因，基于此基因，不仅能反映恶意软件功能上的相似性，也能反映恶意软件之间的借用与传承，即同源性。本文对 19998 个 Android 恶意软件提取了基因，并基于基因对恶意软件进行检测与分类的实验，以验证 Android 恶意软件基因在分析与认知恶意软件中的重要作用。本文的创新点主要有以下几点：

a)定义并提取了 Android 恶意软件基因。恶意软件基因是一种可以分析多种平台恶意软件的概念，可作用于诸如 Windows、Linux、Android 等平台上。同时，Android 平台的恶意软件基因也因 Android 平台的特殊性而与其他平台恶意软件基因有所不同。恶意软件基因可用于分析 Android 恶意软件。

b)构建了基于恶意软件基因的 Android 平台恶意软件检测框架。基于 Android 恶意软件基因，利用支持向量机（SVM）算法搭建了 Android 平台恶意软件检测框架，具有较高的检测效率。

c)提出了新的 Android 恶意软件分类机制。基于软件基因，对 Android 平台的恶意软件进行分类，能够对多种家族的恶意软件起到较好的分类结果。基于不同的训练标签进行分类实验，验证了基因对于分析 Android 恶意软件同源性的有效性。

1 恶意软件基因的定义与提取

1.1 Android 恶意软件基因的定义

为了能够对恶意软件进行系统化地认知与分类，提出了恶意软件基因的概念。首先对软件基因的概念进行定义：

定义 软件基因是软件中携带功能信息的二进制片段。

类似于生物体的基因，软件基因存在于软件体本身，能够表达出软件体的功能与信息，并且在软件的传承中会保存下来，对软件基因的研究可以分析软件的同源性与相似性。基于软件基因的定义，恶意软件基因即是恶意软件中携带功能信息的二进制片段。如同生物基因，恶意软件基因也可以被划分为一些片段。一个恶意软件基因即为能够表达功能信息的最小二进制片段。对于 Android 恶意软件来说，比较关键的恶意基因包括关键恶意性方法、实现关键恶意性方法的编程习惯、以及能够对标志恶意软件及其类别作出贡献的资源文件等。根据这些原则所提取出的恶意软件基因，不仅仅能够表示外在的行为相似性，更能够表现恶意软件之间内在的同源性。其中关键的恶意性方法及其实现习惯来源于 Android 软件的代码段部分，而具有区分性的资源文件则提取于 Android 软件的资源段部分。

1.2 Android 恶意软件基因的提取

对 Android 恶意软件基因的提取，可以分为代码段基因的提取与资源段基因的提取两部分。Android 软件本质上是一个 zip 压缩包，将其解压之后可以得到多个文件及文件夹，其中 classes.dex 即为可执行字节码，亦即本文中的代码段；而 assets 和 res 两个文件夹分别存储了多种类型的资源文件，即为本文中的资源段部分。

1.2.1 提取代码段基因

对于代码段基因的提取，采取了数据流的分析方式。数据流分析是一种常用的基于语义的描述软件的方法，在本文中，为将数据流切分为如同生物基因一样的携带功能信息的片段，将其通过 use-def 链的形式进行提取。所谓 use-def 链，即为软件中数据的从使用到定义的流向，包含一个数据在全部的生命周期中的活动。一个数据的生命周期是从定义开始，到这个数据的重定义或基本块的结束为止。而基本块即是代码段中一段顺序执行的语句序列，一般只有一个入口和一个出口，到发生跳转作为结束。在本文中，采用 Soot 工具对代码段进行分析，

以获取 use-def 链。Soot 作为一个常用的分析 JAVA 程序的工具，现在也可以用于 Android 软件上，Soot 的分析基于几种中间语言，其中在本实验中采用的是 Jimple 语言，这是一种三地址语句，对于分析获取 use-def 链较为方便，算法 1 即为获取 use-def 链的方法。

算法 1

输入：Android apk 文件

输出：Use-def 链

1 遍历 一个 apk 的所有基本块的控制流图:

2 遍历 一个基本块中的每一条语句:

3 获取语句中的定义值和使用值;

4 如果 有以此定义值标记的 use-def 链:

5 输出这条 use-def 链;

6 重新创建一条以此定义值标记的 use-def 链，并将此语句作为链首;

7 否则

8 创建一条以此定义值标记的 use-def 链，并将此语句作为链首;

9 遍历 此语句中的每一个使用值:

10 找到以此使用值标记的 use-def 链;

11 将此语句添加到 use-def 链的末端;

12 结束对一条语句的操作：

13 当此基本块结束时:

14 输出该基本块中当前未结束的所有 use-def 链

在算法 1 中，对使用 Soot 工具分析产生的一种叫做 BriefBodyGraph 的控制流图进行了再分析，以获取 use-def 链。其中，BriefBodyGraph 是一种基于基本块的控制流图，通过对基本块控制流图的分析，可以提取出对所有数据的定义值和使用值（分别名为 def-value 和 use-value）。从第 3 行到第 14 行，通过遍历提取出了一个基本块 use-def 链。当一个值被定义时，一条由此定义值标志的 use-def 链即产生了，而这条链的第一个节点即为这条定义语句，之后每当此值被使用时，使用的语句将被添加到这条 use-def 链的末端。而当此值被重定义时，由此值标志的 use-def 链即被输出，并同时开启另一条由此值标志的 use-def 链。此外，当基本块结束时，此时所产生的所有未结束的 use-def 链都将终止并被输出（即算法一中的 13-14 行）。

为了便于在此后的研究中使用 Android 恶意软件基因进行样本的匹配、比较等工作，对于这些 use-def 链需要进行进一步的形式化表示，将其表示为语句类型加方法调用的形式。作为一种易于分析的三地址语句，Jimple 只有 15 种语句类型，本文中使用单个字母的方式对其中常见的语句类型进行抽象表达，具体类别如表 1 所示。

除了将语句类型进行抽象以外，同时获取了语句中的方法调用，作为基因的重要部分表达语义信息。在 Jimple 语句中，不仅仅调用语句会产生对方法的调用，其他的一些语句类型也会包含方法调用，如定义语句和赋值语句等等。将调用的方法名紧接在调用该方法的语句的类型之后，至此，即获取了代码段的软件基因。如图 1 所示，即为一个软件基因的示例。

表 1　Jimple 语句的抽象表示

语句类型	官方分类	表示字母
定义语句	IdentityStmt	I
赋值语句	AssignStmt	A
调用语句	InvokeStmt	V
空语句	NopStmt	N
判断语句	Ifstmt	F
跳转语句	GotoStmt	G
表转换语句	TableSwitchStmt	S
查找转换语句	LookupSwitchStmt	L
进入监测语句	EnterMonitorStmt	M
退出监测语句	ExitMonitorStmt	W
异常抛出语句	ThrowStmt	T
规范请求返回语句	RetStmt	E
返回语句	ReturnStmt/ReturnVoidStmt	R

```
["I","A","<com.apperhand.common.dto.BaseBrowserItem:
com.apperhand.common.dto.BaseBrowserItem
clone()>","V","<com.apperhand.common.dto.Bookmark: void
<init>(long,java.lang.String,java.lang.String,byte[],com.apperh
and.common.dto.Status)>", "R"]
```

图 1　恶意软件基因示例

这是一个被反病毒引擎广泛命名为 Plankton 家族的恶意软件样本中的一段基因，Plankton 家族的恶意软件收集移动设备上的多种敏感信息并发送至远程服务器上。而这段基因即为收集被感染设备浏览器的书签信息的过程。

通过以上流程，可以获取所有待研究样本的代码段基因，而对于恶意软件基因库的构建，需要对庞杂的基因进行筛选，这部分的工作方法将在 2.2.3 中集中描述。

1.2.2 提取资源段基因

如前文所述，由于 Android 平台软件的特殊性，仅仅通过代码段提取恶意软件基因是不够的，因为资源文件也是 Android 样本中重要的组成部分。本研究中针对不同的资源文件通过不同的方式进行抽象，以将其作为 Android 软件基因。

Android 软件的资源文件的获取方式多样，可以通过 apktool 工具进行反编译，也可以将 Android 软件的后缀名“.apk”改为“.zip”后进行解压操作，这两种方式都将产生 assets 和 res 文件夹，Android 软件的所有资源文件均保存在这两个文件夹中。通过对 Android 软件的分析发现，资源文件主要包括图片文件、配置文件、音频文件、视频文件、文档文件、网页文件等等。以 21 998 个 Android 软件（包含恶意和非恶意软件）作为样本进行分析，发现这些类型的资源文件出现的频率如表 2 所示。

表 2 资源文件的出现频次统计

资源类型	出现频次
配置文件	17897
图片	18889
音频	2678
视频	309
文档	4221
网页	4573

其中，在一个样本中包含一个及以上的某类型资源文件即为出现频次增加一次。由表可知，在 Android 软件中最常出现的资源文件类型是图片文件和配置文件，因此主要针对这两种文件提取基因。

对于图片类型的文件，采取图像指纹的方法对其进行处理。图像指纹是一种成熟的图像分析技术，这是一种对图像进行摘要的方法，而相比于直接获取 MD5 值等 hash 方法，指纹技术能够忽略一些如图片大小不一、个别像素点不同的微小差距，对于相似的图片能够获取一样的摘要值。其具体处理方法为：先将图像处理为统一的大小，再进行灰度处理，并进行去噪操作，最后对剩下的像素点进行摘要，获取图像的指纹。

而对于配置文件，其中最重要的信息在于关键字符串，对 Android 配置文件进行分析，对其出现字符串的前后文进行总结，可通过文本处理的方式获取所有的有效字符串。所谓有效字符串，即去除了表示资源编号、字符大小等无关信息的关键性字符串。

对于其他类型的文件，由于出现频次较低，为便于分析，采用直接计算 MD5 值的方式进行记录。

1.2.3 恶意软件基因的筛选

至此，分别获取了软件代码段和资源段的基因，将此时获得的所有基因称之为“预备基因库”。但对于恶意软件来说，这个预备基因库中包含了大量的冗余数据，可以称之为“无效基因”，如长度较短不具有标志性的代码段基因、在极少数样本中存在的不具有普遍性的基因、以及在恶意软件和正常软件中都会大量存在的不具有恶意性特征的基因等。对于 19 998 个恶意软件进行基因提取后，得到的总基因数据体量超过 10 GB，这不仅对于分析速度和计算机性能是个考验，对于进一步分析结果的准确性也造成了干扰。因此，采用了以下的方法对“预备基因库”进行了去噪处理：

a)删除较短代码段基因。由代码段基因的定义与提取过程可知，代码段基因是由语句序列的类型和调用方法组成的，而较短的且不包含方法调用的语句序列是不具有特殊性的，重复率高且特征性弱，可能出现多种语句序列提取出同样基因的情况，因此，本研究中设定长度不超过 4 条语句且不包含方法调用的基因为较短的无效基因，予以删除。

b)删除低频率基因。在极少量恶意样本中出现的基因不具有普遍性，对于恶意代码的进一步检测、分类等分析难以具备辅助结果，故将删除出现频率较低的基因片段。所谓基因的出现频率，即为包含某一条基因的样本数量与总样本量之比。最终，将出现频率低于 10%的基因予以删除。

c)此时获得的即为恶意代码基础基因库，根据应用场景的不同（检测、分类等），可以通过不同的后续工作搭建不同的应用基因库。将在第三、第四章具体介绍。

2 恶意软件检测框架搭建

利用第二章所述方法，可以得到包含恶意与非恶意软件的基因库，本章将通过支持向量机训练基因库，构造一个检测样本恶意性的分类器，对待测试样本进行恶意性判定。

2.1 构建恶意软件检测基因库

要将恶意性样本与非恶意性样本进行区分，则在训练机器学习模型时不仅仅需要恶意软件基因，也需要非恶意软件基因。因此需要对恶意软件样本和非恶意软件样本分别进行处理，以构建软件基因库。具体处理方式为将恶意样本与非恶意样本都通过第 2 章所述方法提取软件基因，通过去噪处理的前两步，即删除较短代码段基因与低频率基因对其获取的所有软件基因进行预处理。

将预处理完的恶意代码基因与非恶意代码基因进行综合与去重，获取恶意软件检测基因库，将基因库转换为一个长度为 n 的顺序序列，即可以表示为一个 n 维向量$\overrightarrow{Gene}$，向量的第i维度为第i条基因，记为gene_i，即有

$$\overrightarrow{Gene} = (\mathrm{gene}_1, \mathrm{gene}_2, \dots \mathrm{gene}_n)^T$$

2.2 训练支持向量机

对每个训练集样本提取所有的基因，将每个样本的所有基因设为集合S，并为每个样本设置一个 n 维向量$\vec{G} = (\mathrm{g}_1, \mathrm{g}_2, \dots \mathrm{g}_n)^T$，此向量的第i维度$\mathrm{g}_i$为$\overrightarrow{Gene}$中的$\mathrm{gene}_i$是否存在于集合S中，即

$$\mathrm{g}_i = \begin{cases} 1, & if\ \mathrm{gene}_i \in S \\ -1, & if\ \mathrm{gene}_i \notin S \end{cases}$$

在本文中，采用支持向量机作为检测模型的分类器，分别使用恶意样本训练集和非恶意样本训练集对支持向量机的参数进行训练，将获得的参数构成的分类器用于后续对测试集样本的检测。其中本文中使用的支持向量机模型如图 2 所示。

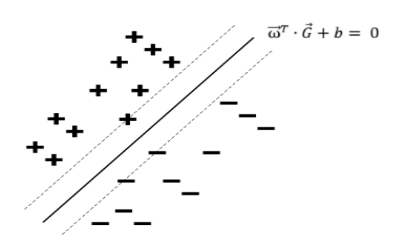


图 2 支持向量机模型

其中有一超平面将样本划分为两部分，分别为恶意样本与非恶意样本，其中超平面所对应的模型为：

$$\mathrm{f}(\vec{G}_i) = \varpi^T \cdot \vec{G} + b$$

其中：$\vec{G}_i$为第i个样本，$f(\vec{G}_i)$为标签，当$f(\vec{G}_i)\geq 1$时，该样本为恶意样本，而当$f(\vec{G}_i)\leq -1$时，该样本为非恶意样本。而$\vec{\omega}^T$和b为支持向量机参数，其中$\vec{\omega}=(\omega_1,\omega_2,\dots\omega_n)$为法向量，而$b$为位移项，分别决定了超平面的方向和超平面到原点的距离。通过训练集样本即可对$\vec{\omega}^T$和b进行训练。

将第i个训练集样本记为$(\vec{G}_i,y_i)$，其中y_i为该样本的标签，即y_i为 1 时代表恶意样本，y_i为-1 时代表非恶意样本。则所有的训练样本集合即为集合$D=\{(\vec{G}_1,y_1),(\vec{G}_2,y_2),\dots(\vec{G}_m,y_m)\}$，通过集合 D 对支持向量机的参数进行训练，使该参数组合构造的支持向量机能够最大化地将训练集样本划分为符合恶意与非恶意样本标签的两部分。将测试集样本输入支持向量机，便可判断其恶意性。

2.3 对测试集样本进行检测

类似于对训练集样本的处理，对于每一个测试集样本，提取其所包含的所有软件基因，再基于恶意软件检测基因库将其向量化，使得每个样本对应一个高维向量$\vec{G}_i$。将这每一个向量$\vec{G}_i$作为输入值计算$f(\vec{G}_i)$的值，当$f(\vec{G}_i)\geq 1$时，该样本为恶意样本，而当$f(\vec{G}_i)\leq -1$时，该样本为非恶意样本。

而作为测试集样本，其本身也具有恶意性标签，以验证本检测模型的准确性。将检测结果标签与参考标签相比较，以在机器学习中常用的准确率和召回率为检验模型准确性的指标。其中，首先设定参数如表 3 所示。

表 3　分类评价参数列表

样本来源	检测结果	
	恶意样本	非恶意样本
恶意样本	TP	FN
非恶意样本	FP	TN

准确率表示所有恶意样本中被正确判断为恶意的比例，而召回率为所有非恶意样本中被正确检测为非恶意样本的比例，其计算方式如下：

$$准确率=\frac{TP}{TP+FP}$$

$$召回率=\frac{TP}{TP+FN}$$

3　恶意软件分类模型构建

基于软件基因的恶意代码研究，根本目的在于基于恶意代码的同源性对其家族特性做分析，因而，在本章中构建了一个恶意软件分类模型，实现对恶意代码的多分类，验证软件基因在对恶意代码同源性研究中所能起到的重要作用。恶意软件分类模型的构建流程如下：

3.1 构建恶意代码分类基因库

在第 2 章搭建的基础基因库的基础上进行进一步操作，获取恶意代码分类基因库：

删除不具有恶意性特征基因。所谓不具有恶意性特征的基因，并非指不具有恶意功能的基因，而是指不能将恶意性样本与非恶意性样本区分出来的基因。实验采取了 2000 个非恶意性样本，采用与提取恶意软件基因同样的方法进行实验，提取非恶意软件基因，将恶意软件基因与非恶意软件基因中高频率重合的部分进行删除。

通过前文的步骤所构建的恶意软件基因库，并不能体现各家族恶意软件的特性，而 10%的频率阈值仍然保留了一些出现频率较低的基因，而之所以采取这个较低的阈值是因为实验的恶意样本中不同家族的比例不一。为了进一步进行恶意软件分类的实验，在分类之前将对各家族的恶意软件基因进行进一步的筛选，删除在本家族中出现频率低于 20%的基因，以构成各家族的基因库。

最后将各家族基因库进行综合，即可得到恶意代码分类基因库。

3.2 获取恶意软件类别标签

在构建恶意代码分类基因库，以及给训练集、测试集样本打标签的过程中，均需要已知恶意代码的家族分类。然而，实验使用的恶意代码样本本身并不具备家族标签，本实验中采取了两种不同的方式为恶意代码样本赋予标签：

3.2.1 使用 AVClass 获取家族标签

AVClass 是由 Sebastián 等人提出的基于反病毒引擎检测结果为恶意代码设置家族标签的系统，它利用了 Virus Total 网站上的反病毒引擎对恶意代码的命名，对一个恶意代码所有命名进行综合分析，提取出各命名中表示家族名称的关键字段，再利用投票系统选出大多数反病毒引擎对这个恶意代码的命名，基于此命名作为家族名。本实验中，根据样本容量将所有家族进行排序，选取样本数量最多的 15 个家族作为下一步分类研究的样本。具体类别数据如表 4 所示。

3.2.2 基于基因进行恶意代码聚类获取标签

另一种给恶意代码样本赋予标签的方法是基于基因对恶意代码进行聚类（聚类方法已在论文[17]中详述），依据恶意代码基础基因库，可以从恶意代码样本中提取向量，提取向量的方法与第三章中相同，将每一个恶意代码抽象为 n 维向量$\vec{G}=(g_1,g_2,\dots g_n)^T$，基于 K-means 方法将所有样本聚类为 15 个类，以自然数 0 到 14 为这 15 个类命名，作为下一步研究中的标签。具体数据如表 5 所示。

3.3 训练多分类支持向量机

类似于第 3 章中的检测框架，本章中的分类框架同样使用基于 SVM 的分类器。传统的 SVM 是一个二分分类器，只能解决“单对单”的问题，即将一个集合的样本划分为正负两类，本文中通过“单对多”的方法构造 SVM 多分类器，即每次将一个家族从其他所有家族从区分出来，通过构造 14 个“单对多”的单分类器，组合起来即为一个能区分 15 个家族的多分类器。

对多分类支持向量机的训练过程也类似于对二分分类器的训练，首先将每个训练集恶意代码抽象为高维向量$\vec{G}=(g_1,g_2,\dots g_n)^T$，而样本标签值设置为 0 到 14 的自然数值，即表

示了 15 个恶意代码家族，即可得到训练集样本D为：

$$D = \{(\vec{G}_2, y_1), (\vec{G}_2, y_2), \ldots (\vec{G}_m, y_m)\}，其中 y_i \in [0, 14] 且 y 为自然数$$

用样本集合D对支持向量机进行训练，即可得到这个多分类器的所有参数，以用于后续的研究中。利用两种不同的标签体系分别对多分类支持向量机进行训练，可得到两个多分类器。

3.4　基于不同标签进行分类测试

基于恶意软件分类基因库，将测试集样本提取基因后全部转换为高维向量。基于前述的两种不同标签体系，分别对测试集样本进行预测，将预测得到的结果与测试集已知的标签进行比对，可分析分类结果的正确性与基因研究的可靠性。

在对分类结果进行分析时引入准确率数组、召回率数组与 A-Jaccard 值三种性能度量数据：其中准确率数组与召回率数组每个家族分类准确率组成的数组。在计算每个家族的准确率时，将本家族视为正样本，而其余所有家族视为负样本，则可计算每一个家族的分类准确率和召回率。而 A-Jaccard 值来源于度量聚类性能的 Jaccard 系数，并稍作修改以适应本文中对多分类的性能度量。假定参考标签将测试集样本划分为 $C=\{C_1, C_2, \ldots, C_k\}$，而分类得到的标签将测试集样本划分为 $C^*=\{C_1^*, C_2^*, \ldots, C_s^*\}$，其中 C_i 与 C_i^* 分别为在两种划分中第i个标签的样本集合。令 λ_m 和 λ_m^* 分别表示样本 m 的参考标签和分类标签。将样本两两配对比较，定义：

$$a = |SS|, SS = \{(x_m, x_n) | \lambda_m = \lambda_n, \lambda_m^* = \lambda_n^*, m < n\},$$

$$b = |SD|, SD = \{(x_m, x_n) | \lambda_m = \lambda_n, \lambda_m^* \neq \lambda_n^*, m < n\},$$

$$c = |DS|, DS = \{(x_m, x_n) | \lambda_m \neq \lambda_n, \lambda_m^* = \lambda_n^*, m < n\},$$

其中SS包含了在C中属于同一家族且在 C^* 中也属于同一家族的样本对，集合SD包含了在C中属于同一家族而在 C^* 中属于不同家族的样本对，集合DS包含了在C中属于不同家族而在 C^* 中属于同一家族的样本对。而如果样本对 (x_m, x_n) 在参考标签和测试标签中均不属于同一个家族，则均认为对于 A-Jaccard 系数不做任何影响。由此可得 A-Jaccard 系数：

$$A - \text{Jaccard} = \frac{a}{a + b + c}$$

而 $A - \text{Jaccard}$ 系数值在[0,1]内，值越大两个反病毒引擎的分类相似性越高。

4　实验结果

4.1　数据集准备

在本文中，采用的样本集合包含了 19 998 个恶意样本和 2 000 个非恶意样本，其中恶意样本收集于以 Virus Share 为主的恶意代码分享的网站与论坛，均在恶意代码检测网站 Virus Total 上可以查询到反病毒引擎查杀结果，而 2 000 个非恶意样本收集自 Android 应用商店的热门榜单，且通过 Virus Total 分析均无恶意性反馈。

4.2　检测模型及验证

基于样本集获取恶意样本基因 4 411 条，非恶意样本基因 2 082 条，为使恶意样本与非恶意样本的样本容量差距较小，以更好地分析检测实验的有效性，检测实验随机抽取了 2 000 个恶意样本和 1000 个非恶意样本作为训练集样本；又另外抽取了 2 000 个恶意样本和 1000 个非恶意样本作为测试集样本。通过测试得到的准确率约为 90.71%，召回率约为 98.37%。与三篇论文中的检测系统[4, 6, 16]进行比对结果如图 3 所示。进一步分析检测结果，可以看到召回率高的原因是对恶意代码的恶意性检测较为准确，而一部分误将非恶意代码标记为恶意代码的样本造成了准确率较低的结果。经分析可知，基于恶意代码基因的分析框架更注重的是恶意代码之间的同源性，而针对同一个正常代码的伪造、重打包代码与原始代码可能存在高度的基因相似性，这也是将一部分非恶意代码误检测的原因。然而，为了保证系统的安全性，防止被恶意代码感染，目前的许多防病毒引擎也同样是以牺牲准确率为代价提高召回率的，同时搭配白名单系统可以有效地在实际应用中防止大部分的对正常软件误报的问题。

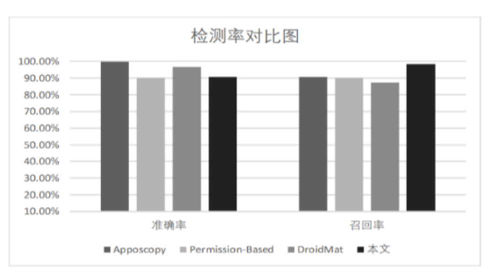


图 3　检测准确率与召回率

4.3　分类模型及验证

在恶意代码分类的实验中，由于 AVClass 将样本集中的恶意代码划分为了 453 个家族，其中有 431 个家族的恶意样本不超过 200 个，相对于总样本容量来说有着数量级的差距，因而难以对所有的家族进行全面性的研究，遂选取了样本容量前 15 的恶意代码家族，其家族名与样本数量见表 5。由于长尾效应，此 15 个家族仅保留了 13 786 个恶意样本。为保证实验的统一性，对变量进行控制，在采取聚类方法获取标签的分类器搭建实验中，也使用了这 13786 个恶意样本。在赋予标签的过程中，将这 13786 个样本通过 K-means 的方法聚类成了 15 类，其各类别样本个数如表 4 和 5 所示。

表 4　基于 AVClass 的恶意代码样本集标签

家族名	样本数量	家族名	样本数量
Admogo	376	Mobwin	308
Adwo	1338	Opfake	1079
Airpush	457	Plankton	331
Dowgin	1751	Smsreg	891
Fakeinst	3914	Umpay	463
Gappusin	1067	Utchi	301
Kuguo	960	Youmi	286
Lotoor	265		

表 5 基于聚类的恶意代码样本集标签

家族名	样本数量	家族名	样本数量
0	4835	8	570
1	582	9	477
2	258	10	1901
3	271	11	225
4	273	12	659
5	412	13	407
6	1721	14	407
7	739		

在这两个分类实验中，分别选取各家族的一半样本作为训练集，另一半样本作为测试集，两个分类实验各家族的准确率和召回率如图 4、5 所示。

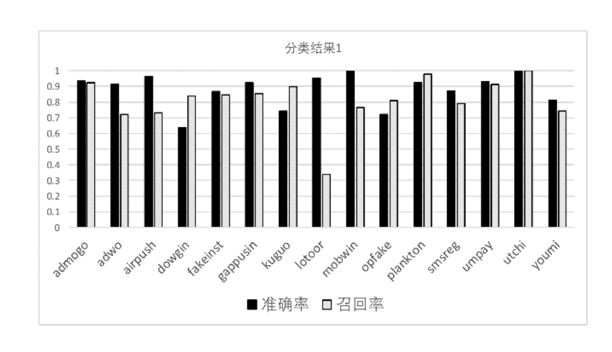


图 4 以 AVClass 标签进行分类的结果

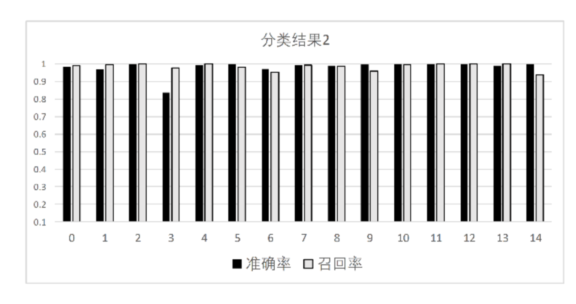


图 5 以聚类标签进行分类的结果

以 AVClass 结果作为标签的分类器的测试 A-Jaccard 值为 0.5344，而以聚类结果作为恶意代码参考标签的分类器的 A-Jaccard 值为 0.9474。

对两个分类器的测试结果的性能度量值进行分析可以发现，基于基因对恶意代码的家族进行研究具有一定的效果。以 AVClass 作为参考标签，其对于个别家族的恶意代码分类效果“不佳”，如 lotoor 家族等。然而，这是因为 AVClass 本身仅仅是一个对标签的投票系统，其受一些应用广泛的反病毒引擎（如 BitDefender）影响较大，对 Virus Total 常用的 100 种反病毒引擎的检测结果进行统计，对本实验中的 Android 恶意代码检测率高于 80%的仅有 24 种，而其中能对 Android 恶意代码进行合理命名的反病毒引擎仅有 20 种，这 20 个反病毒引擎及其对样本的检测率如表 6 所示。

表 6 AVClass 主要反病毒引擎及检测率

反病毒引擎	检测率	反病毒引擎	检测率
NANO	97.5%	GData	91.2%
Ikarus	96.9%	DrWeb	88.8%
ESET-NOD32	96.5%	F-Secure	88.8%
CAT-QuickHeal	96.1%	Ad-Aware	83.7%
Cyren	94.6%	BitDefender	82.6%
F-Prot	92.3%	Kaspersky	81.6%
Sophos	92.3%	Arcabit	81.1%
Avira	92.3%	Emsisoft	80.5%
Fortinet	92.3%	Avast	80.1%
AegisLab	91.4%	eScan	80.0%

其中 Gdata、Ad-Aware、Arcabit、Emsisoft 和 eScan 都使用到了 BitDefender 提供的反病毒引擎，而 F-Prot 和 F-Secure 更是同一家公司的产品，可见 AVClass 赋予的标签受个别引擎影响较大。同时，由表 6 可见只有检测率最高的反病毒引擎 NANO 和基于基因的检测系统召回率相近，由此也可证明基于基因的 Android 恶意软件检测系统的有效性。

虽然基于基因的研究方法与这些传统的反病毒引擎具有一定的差异性，但从分类结果上大面积的重合也可以佐证基因对于恶意软件分类分析的有效性。而基于聚类结果的参考标签则与分类结果基本一致，可以验证基于基因的分类能够基本契合聚类结果，通过软件基因对恶意软件的研究是具有稳定性、可靠性的。

5 结束语

本文创造性地提出了 Android 平台的恶意软件基因，并给出了系统化的自动化提取软件基因的方法，通过软件基因搭建了恶意软件检测框架与分类框架，都起到了良好的效果，并能够对软件基因在恶意软件同源性分析上的有效性提供佐证。在下一步工作中，拟基于恶意软件基因提出 Android 平台恶意软件系统化命名方案，以改善目前研究中对恶意软件命名与标志混乱而导致的不利于科学研究及业界分析的现状。

参考文献：

[1] Egham. Garther says worldwide smartphone sales grew 3. 9 percent in first quarter of 2016 [EB/OL]. [2016-05-19]. http://www. gartner. com/newsroom/id/3323017.

[2] Alibaba. 2015 annual report on mobile security viruses [EB/OL]. [2016-02-23]. http://jaq. alibaba. com/community/art/show?spm=a313e. 7768735. 8000000. 21. qXFA8r&articleid=193.

[3] Enck W, Ongtang M, McDaniel P. On lightweight mobile phone application certification [C]// Proc of ACM Conference on Computer and Communications Security. 2009: 235-245.

[4] Aung Z, Zaw W. Permission-based Android malware detection [J].

International Journal of Scientific & Technology Research, 2013, 2 (3): 228-234.

[5] Fuchs A P, Chaudhuri A, Foster J S. SCanDroid: automated security certification of Android applications [R/OL]. University of Maryland, http://www. cs. umd. edu/~avik/projects/scandroidascaa. pdf, 2009.

[6] Wu Dongjie, Mao Chinghao, Wei Te'en, *et al.* DroidMat: Android malware detection through manifest and API calls tracing [C]// Information Security. 2012: 62-69.

[7] Chan P P F, Hui L C K, Yiu S M. DroidChecker: analyzing android applications for capability leak [C]// Proc of ACM Conference on Security and Privacy in Wireless and Mobile Networks. 2012: 125-136.

[8] Lu Long, Li Zhichun, Wu Zhenyu, *et al.* CHEX: statically vetting Android apps for component hijacking vulnerabilities [C]// Proc of ACM Conference on Computer and Communications Security. 2012: 229-240.

[9] Kim J, Yoon Y, Yi K, Shin J. ScanDal: Static analyzer for detecting privacy leaks in Android applications [J]. Mobile Security Technologies, 2012 (12) .

[10] 李挺, 董航, 袁春阳, 等. 基于 Dalvik 指令的 Android 恶意代码特征描述及验证 [J]. 计算机研究与发展, 2014, 51 (7): 1458-1466.

[11] Cui Xingmin, Yu D, Chan P, *et al.* CoChecker: Detecting Capability and Sensitive Data Leaks from Component Chains in Android [C]// Information Security and Privacy. [S. l.] : Springer International Publishing, 2014: 446-453.

[12] Feng Y, Anand S, Dillig I, *et al.* Apposcopy: semantics-based detection of Android malware through static analysis [C]// Proc of ACM Sigsoft International Symposium on Foundations of Software Engineering. 2014: 576-587.

[13] Rastogi V, Chen Y, Jiang X. DroidChameleon: evaluating Android anti-malware against transformation attacks [C]// Proc of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security. 2013: 329-334.

[14] Zhou Yajin, Jiang Xuxian. Dissecting Android Malware Characterization and Evolution [C]// Proc of IEEE Symposium on Security & Privacy. 2012: 95-109.

[15] Sebastián M, Rivera R, Kotzias P, *et al.* AVclass: a tool for massive malware labeling [C]// Proc of International Symposium on Research in Attacks, Intrusions, and Defenses. 2016: 230-253.

[16] Feng Y, Anand S, Dillig I, *et al.* Apposcopy: semantics-based detection of Android malware through static analysis [C]// Proc of ACM SIGSOFT International Symposium on Foundations of Software Engineering. 2014: 576-587.

[17] Han Jin, Zhao Rongcai, Shan Zhen, *et al.* Analyzing and recognizing android malware via semantic-based malware gene [C]// Proc of the 9th International Conference on Cyber-enabled Distributed Computing and Knowledge Discovery. 2017: 17-29.